Obtaining the Greatest Scientific Benefit from Observational Platforms by Consideration of the Relative Benefit of Observations

David Chelberg, Frank Drews, David Fleeman, and Lonnie Welch

Center for Intelligent, Distributed and Dependable Systems
School of Electrical Engineering and Computer Science
Ohio University, Athens, Ohio – 45701
(chelberg| drews | welch @ohio.edu)

Jane Marquart, Barbara Pfarr

Real-Time Software Engineering Branch
NASA Goddard Space Flight Center
Baltimore, Maryland -20771
(jane.marquart | barbara.pfarr@gsfc.nasa.gov)

Abstract

One of the current trends in spacecraft software design is to increase the autonomy of onboard flight and science software. This is especially true when real-time observations may affect the observation schedule of a mission. For many science missions, such as those conducted by the Swift Burst Alert Telescope, the ability of the spacecraft to autonomously respond in real-time to unpredicted science events is crucial for mission success. We apply utility theory within resource management middleware to optimize the real-time performance of application software and achieve maximum system level benefit. We then explore how this methodology can be extended to manage both software and observational resources onboard a spacecraft to achieve the best possible observations.[1]

1. Introduction

One of the current trends in spacecraft software design is to increase autonomy and automation in onboard flight and science software. This is especially true when real-time observations may affect the observation schedule of a mission. One example of this is the Swift Burst Alert Telescope instrument [15]. Swift will respond to onboard detection of gamma ray bursts in real-time. Its actions will include: notifying the ground of the event via TDRSS; interrupting the planned science schedule and commanding the spacecraft to reorient its attitude toward the burst; reconfiguring the onboard processor for Gamma Ray Burst (GRB) data processing; and intercepting the onboard science plan following completion of the burst. The decision of when and whether to interrupt the on-going survey task and respond to a particular GRB event is a function of the relative merit of each proposed observation. When the scientific benefit of responding to a perceived GRB event is greater than the scientific benefit of continuing the survey, the spacecraft autonomously decides to reorient and respond to the GRB event. The ability of the spacecraft to autonomously respond in real-time to unpredicted science events is crucial for this mission's success.

This example demonstrates the use of increasingly powerful computers to perform science processing and analysis in real-time on-orbit. As this trend continues, spacecraft will need to make increasingly complex autonomous decisions about which action among several will return the most valuable scientific data. This task is even more complex when spacecraft are involved in coordinated scientific observation involving constellations of spacecraft and multiple instrument platforms as NASA envisions for future missions.

Utility theory, the theory that provides mathematical tools to compute the relative value of different courses of action, was originally developed by economists to model the decisions made by consumers [8]. It has been addressed by many researchers and used in diverse fields to determine optimal decisions, i.e., to decide which action among several is the best one [9], [10] (compare [11] for an overview). It has been recently incorporated in adaptive resource management middleware (ARM) that optimizes the real-time performance of sets of application software. The middleware plans actions that include which software to run on which resources to achieve the maximum system level benefit. Our previous research in the field of ARM includes static models for resource allocation of real-time systems [1, 2] and dynamic models in [3, 4]. Applications of our dynamic models [5, 6, 7]

[1] This work was funded in part by the NASA Earth Science Technology Office Advanced Information Systems Technology Program; by the NASA Computing, Information and Communications Technology Program; and by the DARPA Program Composition for Embedded Systems Initiative.

showed their effectiveness for adaptive resource management. However, our previous approaches lacked the information needed to gracefully degrade performance in overload situations, did not support feasibility analysis or allocation optimization, did not consider security aspects, and did not include network hardware. To overcome these drawbacks, we proposed a general optimization framework for distributed, dynamic real-time systems [12]. Interesting aspects of this model include dynamic environments, and utility and service levels, which provide a means for graceful degradation in resource constrained situations.

In [13] we proposed a hierarchical ARM architecture based on our general framework and a heuristic algorithmic approach based on a table lookup technique to solve the corresponding resource allocation problem. This paper explores how this methodology can be extended to manage observational resources in addition to the software resources onboard a spacecraft to achieve the best possible observations.

Chapter 2 presents an overview on the RM system framework presented in [12] and [13]. Chapter 3 proposes an architecture for an adaptive resource manager. Chapter 4 describes how our model can be used to incorporate the relative benefit of observational resources. Chapter 5 provides an example describing how our framework would act onboard a satellite.

2. RM System Framework (Overview)

In this section we summarize the general resource management model presented in [12] and [13]. The model assumes that available CPUs can be treated as a (possibly heterogenous) set of computational resources that software tasks are run on. It is possible to run a task on a choice of CPUs, thereby requiring the RM system to allocate tasks to CPUs. This also allows the system to have greater fault tolerance. If a particular CPU is down, the tasks may be reassigned to other processors. This framework is applicable in various contexts such as the total shipboard computing environment for warships, cooperating robots, onboard satellite systems, and for many ground-based real-time systems. It is our goal to remain context-free in describing the model, while showing the applicability to the satellite domain in subsequent sections.

A dynamic real-time system is composed of a variety of software components, as well as a variety of physical (hardware) components that govern the real-time performance.

The physical components of a real-time system can be described by a set of computational resources and an interconnection network, and other devices. The computational resources are a set of host computers $H = \{h_1, \ldots, h_m\}$. It is generally assumed that the properties of the computational resources and the network resources are known.

Regarding the software components, we distinguish two different software systems (see [12]):

- The application software system. Its general purpose is to control the environment. It may also include the control of hardware components of the real-time system. There is a logical view that distinguishes several levels of abstraction. The highest level represents the whole application software system. This is often referred to as "system", which, on the next lower level, is considered as a collection of subsystems $SS_1, \ldots, SS_m$
- The surveillance and management system (or Resource Manager, RM) is responsible for the correct cooperation between the components of the application software, to ensure real-time conditions such as timeliness, dependability, security, etc.

A subsystem represents some part that can be logically separated from the total system. It is responsible for managing a part of the real-time system, such as a hardware component. A subsystem is a collection of paths, each with a given period or a given maximum frequency. Each path consists of a set of tasks and possible precedences between them. Tasks inherit the period or maximum frequency from their paths.

Scheduling theory does not only deal with task-to-processor allocation, but also with schedules. Besides task allocation, a schedule defines the start times of tasks. For a periodic task we may simplify the schedule specification by a greedy rule, saying that each time a host becomes free it chooses a task of highest priority among all tasks that are waiting to be serviced by the host processor. Priority may be defined externally. Other common ways are by deadline (earliest deadline first strategy, EDF), or by the rate monotonic rule. The event-driven tasks with given rate r can be treated as periodic tasks with period r^{-1}. By specifying the scheduling rule for the execution order, the schedule is uniquely determined from the task allocation. As we will see, only the feasibility of an allocation has to be checked.

During their execution the tasks will require certain system resources. In particular it is assumed that tasks have known execution and memory profiles that depend on various circumstances,

- on the processor it is assigned to,
- on certain parameters that cannot be changed ("extrinsic parameters"); these define operational conditions and are set by the environment or by the system components.
- on service parameters; they can be changed at runtime, e.g., in order to optimize system benefit.

See [12] for a detailed description.

The execution of tasks, and consequently the system behavior is highly determined by the values of a number of attributes. There are two types: extrinsic attributes $E = (e_1, ..., e_k)$, and service attributes $S = (s_1,...,s_l)$. Extrinsic attributes express functional conditions or requirements. Extrinsic attributes are either posed by the environment, or by the status of the system components. They are set by external conditions and cannot be changed by the system. It is generally assumed that the values of an extrinsic attribute ranges within a known interval [min, max], where higher values represent more stringent conditions.

An example of an extrinsic attribute dictated by the environment "externally" is the period at which a controlling action has to be performed. An example of an "internal" extrinsic attribute, defined by the system, is the availability of processors, buffers, or internal network bandwidth.

Each subsystem SS_i has to react properly upon changes of extrinsic attributes. We denote by E_i the subset of extrinsic attributes that must be handled by SS_i.

In contrast to the extrinsic attributes, the service attributes $S = (s_1, ..., s_l)$ are entities that can be changed at any time by the controlling system. If the external conditions (represented by the extrinsic attributes) change, appropriate settings of the service attributes allow adaptation to the new conditions.

Given an extrinsic attribute with range [min, max], we expect consequences for the system behavior only if the attribute changes by at least some minimum amount. Therefore we discretize attribute values by defining a mapping of the interval [min, max] to a finite set of discrete values. There is no loss of generality if we choose a set of integers $0, 1, ..., max\_e$, where max_e is defined by an acceptable granularity. The mapping $\gamma: [min, max] \rightarrow \{0, 1, ..., max\_e\}$ is defined by $\gamma(a) = \lfloor (a - min) \cdot \frac{max\_e}{max - min} + 0.5 \rfloor$.

Henceforth we assume that, along with k extrinsic attributes $e_1, ..., e_k$, there are given maximum integers $max\_\tilde{e}_1, ..., max\_\tilde{e}_k$. There are mappings of extrinsic attribute values to grid points,

$$\gamma_i : [min\_e_i, max\_e_i] \rightarrow \{0, ..., max\_\tilde{e}_i\} \ (i = 1, ..., k).$$

These values define a grid of dimension k, with $max\_\tilde{e}_i + 1$ points in direction i,

$$G = \{ (\tilde{e}_1, ..., \tilde{e}_k) \mid \tilde{e}_i \in \{0, ..., max\_\tilde{e}_i, i = 1, ..., k \}.$$

Similarly to the external attributes, we also discretize service attributes. If the range of service attribute s_i is $[min\_s_i, max\_s_i]$ (discrete or continuous), then a mapping of service attribute s_i to a discrete service attribute $\tilde{s}_i$ is defined by $\delta_i : [min\_s_i, max\_s_i] \rightarrow \{0, ..., max\_\tilde{s}_i\}$. This granularization leads to discrete service attributes $\tilde{s}_i$ with range $\{0, ..., max\_\tilde{s}_i\}$.

For both, extrinsic attributes and service attributes, we use tilde (~) to denote discretization; e.g., the vector of discrete service attributes is denoted by $\tilde{S} = (\tilde{s}_1, ..., \tilde{s}_l)$.

In both, the pre-runtime analysis and the runtime system, we consider only discretized values of extrinsic and service attributes. For simplicity, the terms "extrinsic attribute" and "service attribute" are used in the discrete sense.

Finding optimal allocations is generally a hard problem, consuming time that is at least exponentially increasing with the number of tasks. Checking the feasibility of a chosen allocation, however, can be done very fast and in time that increases linearly with the number of tasks. There are many ways to obtain sub-optimal allocations, for example by applying heuristic scheduling strategies. In these, a number of allocations is chosen in a way specified by the heuristics, and checked for feasibility and optimality.

In [13] we followed an approach in which a set of allocations $\{alloc_1, ..., alloc_K\}$ is chosen heuristically, and each is analyzed with regard to its feasibility under different settings of extrinsic and service attributes. Since we perform our analysis on discrete attribute values, for each allocation $alloc_i$ it is checked on which grid points it is feasible. Maximum points are referred to as Pareto points. For an allocation $alloc_j$ $(j \in \{1, ..., K\})$, and a vector $\tilde{S}$ of discretized service attribute values, $Pareto_{\tilde{S}}(alloc_j)$ denotes the set of grid points $\tilde{E} = (\tilde{e}_1, ..., \tilde{e}_k)$ that have the properties

(i) $alloc_j$ is feasible if the service attribute values are set to $\tilde{S} = (\tilde{s}_1, ..., \tilde{s}_l)$

(ii) there is no point $\tilde{E}' = (\tilde{e}_1', ..., \tilde{e}_k') \in Pareto_{\tilde{S}}(alloc_j)$, $\tilde{E} \neq \tilde{E}'$, that dominates $\tilde{E}$ (i.e., $\tilde{e}_i' \geq \tilde{e}_i$ for $i = 1, ..., k$).

The set of Pareto points stands for a set of allocations along with settings of service level attributes that allow a maximum increase in extrinsic attributes. Hence, this set represents an investigation of the operational limits of the real-time system.

In [13] an algorithm is presented that determines Pareto points during a pre-runtime analysis.

For local optimization in the subsystem SS_i, a function $U_i(\tilde{S}_i, \tilde{E}_i)$ called "local utility function" is provided. The

objective is to set the service attributes such that the system operates optimally, i.e. a maximum profit is gained. To measure the profit, an overall utility function U_{System} is defined whose value depends on both the service and the extrinsic attributes. This allows one to evaluate each point of the $Pareto_{(\tilde{s}_1, \ldots, \tilde{s}_l)}(alloc_i)$:

$U_{System} : ((\tilde{s}_1, \ldots, \tilde{s}_l), G) \rightarrow IR$ (real numbers)

Function U_{System} gives a numerical benefit that is attained if the extrinsic attribute values are G, and the system service attributes s_i are set to $\tilde{s}_i$ ($i = 1, \ldots, l$). A practical way is to compute U_{System} from the subsystem utilities by means of some aggregation function: U_{System} := aggregation($U_1, \ldots, U_m$). For example, for numerical weights $w_1, \ldots, w_m$ we may define

$$U_{System} := \sum_{1}^{m} w_i U_i$$

The objective of the pre-runtime analysis is to investigate operational limits of the control system. Given sets of extrinsic and service attributes, we want to find a feasible allocation that maximizes the values of extrinsic attributes, while the values of the service attributes are such that an overall utility is maximized. Indeed we are facing a high-dimensional multicriterion optimization problem, which, due to its computational complexity, can only be solved heuristically (cf. [13]).

For each of $K > 0$ chosen allocations $\{alloc_1, \ldots, alloc_K\}$, Pareto-points are determined. Each set of Pareto points $Pareto_{\tilde{S}}(alloc_i)$ can be represented by a service table

$$ST(alloc_i) = \{ \langle \tilde{S}, \tilde{E} \rangle \mid \tilde{E} \in Pareto_{\tilde{S}}(alloc_i) \text{, and } \tilde{S} \in \{0..max\_\tilde{s}_1, \ldots, 0..max\_\tilde{s}_l\} \} .$$

At runtime, the allocation manager chooses an allocation by selecting a service table that best fits current requirements of extrinsic attribute values (compare [13] for more details).

3. ARM Architecture

The A*RM* is responsible for the correct operation of the whole system. As input, it is given the static characteristics of both the hardware system and the software system. Based on these, it makes resource allocation decisions and has the ability to modify certain performance parameters such as service attributes. The resource manager consists of an allocation manager (*AM*) which chooses a new allocation of application software to hosts due to major changes of extrinsic requirements, a global meta agent (*GMA*) which checks if reallocation of application software to hosts is necessary, and tries to optimize total benefit, meta agents (MA_i, $i = 1, \ldots, m$) each being responsible for controlling an application subsystem (SS_i).

One reason for including *GMA* is overall utility: *each* MA_i tries to optimize its own behavior. In contrast, *GMA* has a global system view that allows for an optimization on a higher level through negotiation. Another reason is efficiency: a reallocation of software modules among hosts involves major changes that take considerable time and affects the real-time requirements of the environment. Hence, the attempt is to delay reallocations as long as possible, and instead to make orders of magnitude faster adjustments of system parameters (the service attribute settings) on the *GMA* level.

3.1. Allocation Manager

One of the main objectives is to find an optimal allocation of the applications to host computers. Such an allocation, formally described by a function *host:* $A \rightarrow H$, has to fulfill runtime conditions and memory limitations on the hosts. Both execution time and memory usage of a task depend not only on extrinsic and service attribute parameters, but also on the machine on which the task is being executed.

If the actual instantaneous extrinsic attribute values change and exceed the operational limits of the current allocation, the *GMA* triggers the *AM* to choose and install a new allocation. The *AM* will then decide upon a new allocation that is able to handle the requirements of the new extrinsic values. The decision is based on the global service tables $ST(alloc_1), \ldots, ST(alloc_K)$ provided by the pre-runtime analysis, and on the utility that potentially can be gained from the new allocation.

A concept that offers the ability of *adaptation* to trends in the development of extrinsic attribute values, is the use of statistical data about extrinsic attributes. Assume, for example, that for each extrinsic attribute e_i, both, a mean value $\bar{e}_i = \gamma_i(\bar{e}_i)$ and standard deviation $\sigma_i := \sigma(\bar{e}_i)$ are available. $\bar{E}^{new}$ and σ^{new} can be computed from the last N changes of extrinsic values. N can be interpreted as the length of a sliding window that specifies how far historic values are considered. For example, for $N = 1$ we get $E^{new} = \bar{E}^{new}$ and $\sigma^{new} = 0$. In this case, the model becomes deterministic because the algorithm must choose an allocation that is able to dominate the new requirement E^{new}. If $N > 1$, the algorithm will accept an allocation that potentially violates domination with some probability, i.e., also non-dominating allocations are accepted.

The advantage of this approach is two-fold:

- Requirement peaks can be ironed away.
- The size of history determines how fast the control mechanism is able to adjust to new requirements.

In [13] the algorithm to realise this strategy is explained in more detail.

A component-wise defined function $f(\bar{E}^{new}, \sigma^{new})$ is used to set minimum requirements for the manageable extrinsic values. An example function is $f(\bar{E}^{new}, \sigma^{new}) := \bar{E}^{new} - \sigma^{new}$. Considering $f = (f_1, \ldots, f_m)$ as a point in the k-dimensional space of extrinsic values, we measure the distances between Pareto points (P) and f by applying a metric that gives extrinsic attributes with a smaller standard deviation a higher influence on the distance,

$$weighted\_distance\ (P,f) := \sqrt{\sum((p_i - f_i)/\sigma_i)^2}\ . \quad (1)$$

The adaptable strategy chooses a Pareto point P that dominates f and has smallest weighted distance to f.

With knowledge of f, the first step of the AM is to remove all entries from each $ST(alloc_i)$ that do not dominate f. The result is a set of K reduced service tables. In the next step, the AM computes the utility for each entry of each reduced table. Among these reduced tables the one with the highest utility entry is kept and sent to the GMA, and its associated allocation is realized (arbitration is applied in case of ambiguities). The component $\tilde{Em} = (em_1, \ldots, em_k)$ of this table is called *manageable* extrinsic attribute vector, because it defines the maximum possible extrinsic values for each service setting that can be handled by the new allocation.

In the computation of U_i, the requirement vector $\tilde{E}_i$ is used instead of $\tilde{Em}_i$ for the following reason: though the chosen allocation offers more capabilities (expressed by the dominating $\tilde{Em}_i$ vector), the system will not realize higher requirements than necessary (expressed by $\tilde{E}_i$).

3.2. Global Meta Agent

The purpose of the GMA is to provide an interface between the allocation manager and the Meta Agent MA_i of each subsystem.

The GMA essentially consists of two algorithms that are described in [13].

Note that in this solution, only one tuple is returned to each MA_i receives exactly one vector $\langle \tilde{S}_i\ ,\ \tilde{Em}_i \rangle$. To assign MA_i higher functionality (and hence more responsibility), GMA can return a set of possible tuples, and give the MA_i's freedom to choose a setting according to their needs.

The algorithm *Reallocation_Request* checks if a call to the allocation manager is required, or if an adjustment without allocation change is possible (cf. [13]).

Feasibility of an allocation is guaranteed as long as there is a table entry in $ST(alloc)$ whose manageable extrinsic attributes dominate the given extrinsic attributes G. In case of a request from MA_i, the GMA (i.e. the reallocation-request) quickly has to decide if a new allocation is requred. Alternatively, the reallocation_request may take into account, how many dominating tuples remain in $ST(alloc)$, and triggers the allocation manager if this number becomes too small. Another criterion can be utility-oriented: if the total utility drops more than some minimum amount, then a new allocation is requested.

3.3. Meta Agents

The concept of a separate Meta Agent MA_i for each subsystem SS_i makes it possible to physically distribute the meta agent software components. The realisation of a local optimizer algorithm is explained in [13].

Among those service table entries (identified by the partial service attribute values of $\tilde{S}_i$ for which the allocation is feasible), choose one with maximum local utility U_i. This way service attribute settings are chosen such that utilities are maximized in each subsystem.

If the extrinsic attributes change during runtime, the local optimizer in the subsystem Meta Agent MA_i adjusts to these changes by first checking (local) feasibility of the entries of the partial service table. As a consequence MA_i might find a new set of service attribute settings, subject to optimizing the subsystem utility. In this case it sends suggested partial service attribute values as a request to the Global Administrator.

3.4. Cooperation between the components of the RM

The situation defined by current extrinsic attribute values and service attribute settings changes if the extrinsic values change. In this case, the MA_i's check if the altered settings of service attributes would allow improved local utilities.

If a meta agent MA_i realizes that its local utility can be raised by some minimum amount, it issues a request to the GMA to change service attribute settings. The GMA

responds to this request by checking the service table *ST(alloc)*. If it finds an entry $(\tilde{S}, \tilde{Em}, U)$ that improves overall utility, while the extrinsic attributes are dominated by $\tilde{Em}$, the *GMA* would return $\tilde{S}$ and the new manageable bounds $\tilde{Em}$ to all MA_i's . If there is no such entry in the service table, then the changes in extrinsic attribute values cannot be handled by the current allocation *alloc*, and the allocation manager is triggered to choose a new allocation that is better adjusted to the current situation.

A direct response to the MA_i's is expected to need orders of magnitude less time than asking the allocation manager for a new allocation. Therefore the *GMA* is designed such that it tries to keep the current allocation as long as possible.

The data structure based on which the *GMA* makes its changes is the service table *ST(alloc)*. As a consequence, changes in the extrinsic attribute values excludes those table entries from consideration, whose $\tilde{Em}$-components do not dominate the new extrinsic setting $f(\bar{G}^{new}, \sigma^{new})$. Criteria for calling the allocation manager *AM* are: The *ST(alloc)* is exhausted, i.e. it has no entries dominating $f(\bar{G}^{new}, \sigma^{new})$. A weaker condition is utility-oriented: If utility drops considerably (by some specified maximum amount), or it drops below some given limit, then, though there might still be dominating entries in *ST(alloc)*, the *AM* is called.

4. Modelling Observational Resources

Observational spacecraft software is often driven by a schedule determined a priori. Such a schedule may indicate to a sensor when to start an observation, the duration of an observation, the rate of receiving data, and the importance to the scientist. We claim that the information stored in such observation schedules can be modeled in our framework as extrinsic attributes and used by ARM control software to plan for better utilization of the spacecraft resources. In addition, ARM can effectively react to dynamic changes in observation schedules due to unpredicted science events such as Gamma Ray Bursts (GRB) in the Swift Burst Alert Telescope [15] system. In [13], an Image Processing System (IPS) is introduced as possible observation-based spacecraft software. An observational schedule for such a system may appear as in figure {1}. In this example, the rate that the camera will take pictures during a time frame can be determined by the extrinsic attribute period. The ARM control software must provide a resource allocation that allows the IPS to meet all real-time constraints for the current observation while maintaining optimality of system utility. In addition, ARM must react to schedule changes such as the sudden increase in importance due to the onboard detection of hurricane formation on the Atlantic Coast.

```
...
<observation name="atlanticcoast">
  <time name="start">20030317 03:00:00</time>
  <time name="end">20030317 03:30:00</time>
  <extrinsic name="period" unit="sec">30</extrinsic>
  <extrinsic name="importance">5</extrinsic>
  <extrinsic name="cloudcover" >70</extrinsic>
</observation>
...
<observation name="washingtondc">
  <time name="start">20030317 04:30:00</time>
  <time name="end">20030317 05:00:00</time>
  <extrinsic name="period" unit="sec">30</extrinsic>
  <extrinsic name="importance">10</extrinsic>
  <extrinsic name="cloudcover" >45</extrinsic>
</observation>
...
```

Figure 1: Example Observation Schedule

Another important reason for having a framework to model observation schedules for spacecrafts is due to the presence of persistent (nonrenewable) resources and constrained downlink availability and capacity. With such a framework, ARM software can also plan and control the usage of persistent resources such as onboard storage buffers to prevent less important data from excluding more important data in the presence of resource overloads. In some sense, the utility of a spacecraft mission is directly proportional to the quality of data that is returned to the ground. The observation schedule can be used by ARM to determine the expected storage requirements for a subsystem. These expected requirements can be used as a basis for decisions concerning the use of the persistent resources to optimize the utility of the data that eventually reaches the scientist. Much work remains to be done in developing ARM policies for persistent resources.

5. Satellite Example

The Swift [15] system for identifying and processing Gamma Ray Bursts (GRB) is a current NASA mission that requires autonomous decisions to be made onboard satellites. In Swift, a sensor scans space looking for a GRB. Once a GRB is detected, the satellite determines the originating direction of the GRB, slews the sensor towards the GRB source, and observes the afterglow produced by the GRB. The current Swift implementation does not use resource management middleware to accomplish these dynamic tasks. However, in this section,

we describe how such a satellite system could operate under the control of the proposed resource management framework.

The Swift mission can be decomposed into the following tasks: scanning by the Burst Alert Telescope (BAT), detection by the BAT, slewing by the satellite positioning system, observation by the X-Ray Telescope (XRT), and observation by the Ultra-Violet/Optical Telescope (UVOT). Other potential tasks that may need to be performed simultaneously include the collection of health and safety data and power management.

The Swift system currently operates without resource management (RM) middleware as follows. All three instruments (BAT, XRT, and UVOT) continuously produce images in the direction they are pointing based on some positioning schedule for surveying/scanning that is created a priori. In the presence of a new GRB source, Swift autonomously updates its observation schedule so the system can capture images of the new GRB. Since GRB images are the most informative during the first few minutes, Swift attempts to capture new GRB sources as soon as possible. Once a set amount of images of a GRB source have been collected, Swift will again follow the positioning schedule for surveying/scanning.

The same functionality described in Swift can also be conducted within the presence of resource management (RM) middleware. If RM were to be used for controlling the Swift system, then the various RM components would also be running on the satellite. Resource monitors would gather the current state of all onboard resources. Resource control mechanisms would be in place to allow the relocation of applications to different processors and various other control techniques. In addition, the components that realize the decision-making within our proposed framework would also be running on the satellite: the Allocation Manager, the Global Meta Agent, and a Meta Agent for each subsystem.

In an RM-compliant implementation of Swift, the positioning schedule could be modeled as a set of service attributes to be given to the satellite positioning agent, where possible attributes include the location, duration, and relative scientific importance. In the absence of a new GRB, the Meta Agent would determine the service attributes for positioning based on the predefined positioning schedule, and send the current desired position to the positioning agent. When new GRB sources have been detected, the Meta Agent would be responsible for deciding when images of the new sources should be collected. If there is only one new GRB source, the Meta-Agent would immediately reposition to gather images of the GRB source since GRB images are most informative immediately following detection. If more than one new GRB source occurs within a few minutes of each other, the Meta Agent must consider the tradeoffs between repositioning the satellite for the second GRB source. This decision would be influenced by utility models that take into account the preferences of scientists for such a situation. These models may also take into account specific extrinsic attributes concerning a GRB image such as gamma-ray count rate and redshift.

RM could also allow Swift processes to vary the resource consumption of each instrument and application. The rates at which the BAT, XRT, and UVOT perform their operations could be modeled as service attributes that RM could control resulting in different operating levels for each instrument. If storage space for images on the satellite is constrained, then the rate at which the Swift instruments create images could be changed possibly resulting in fewer images. This action would be triggered only in the case that on-board storage was insufficient to keep all data.

In the absence of a new GRB, it may be desirable for the XRT and UVOT tasks to use a nominal amount of resources since they are not gathering information about a GRB. The absence of a GRB could be modeled as an extrinsic attribute that can assume two values indicating whether a GRB is present or not. The BAT would continue to scan and report to a Meta Agent that a GRB has not been detected, while the XRT and UVOT continue to perform a nominal sky survey. This allows other onboard applications to increase their resource consumption to perform various tasks such as further process the images collected thus far. Once a GRB has been detected, the BAT would reset the extrinsic attribute to indicate that a GRB has been detected and report it to the Meta Agent. The Meta Agent would notify the Global Meta Agent that a GRB is present and that it would like to have more resources for the XRT and UVOT tasks. The Global Meta Agent may ask the Allocation Manager for a new allocation or it may readjust the allowable service attributes for each subsystem. Other tasks on the satellite could either be shutdown or could perform at reduced service levels while the XRT and UVOT are gathering and processing information about the current GRB.

The discussion thus far concentrates on a single satellite system. The Swift project will demonstrate that individual autonomous satellite systems can be successfully deployed without resource management middleware. However, as NASA seeks to transition to constellations of cooperative satellites requiring autonomy, the need arises for a consistent framework for resource management. We have discussed how our resource management framework could be applied to the Swift system with the realization that the same RM framework could be used to control the processing within a constellation of satellites.

6. Conclusion

In this paper, we have described in greater detail our previous model [12] for characterizing distributed real-time systems operating in dynamic environments. We have also explored the applicability of such an adaptive resource management framework for increasing autonomy in spacecraft flight and science software. The main contribution of this paper is to describe how to apply our model to manage both software and observational resources onboard a spacecraft to optimize the relative benefit of observations.

7. References

[1] Verhoosel, J., et al., "A Model for Scheduling of Object-Based, Distributed Real-Time Systems," *Journal of Real-Time Systems*, 8(1), pp. 5-34, Kluwer Academic Publishers, January 1995.

[2] Welch, L., Stoyenko, A., and Marlowe, T., "Modeling Resource Contention Among Distributed Periodic Processes Specified in CaRT-Spec," *Control Engineering Practice*, 3(5), pp. 651-664, May 1995.

[3] Welch, L., et al, "Adaptive QoS and Resource Management Using A Posteriori Workload Characterizations," *The IEEE Real-Time Technology and Applications Symposium*, pp. 266-275, June 1999.

[4] Welch, L., et al, "Specification and Modeling Of Dynamic, Distributed Real-Time Systems," *The IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, pp. 72-81, December 1998.

[5] Welch, L. and Shirazi, B., "A Dynamic Real-Time Benchmark for Assessment of QoS and Resource Management Technology," *The IEEE Real-Time Technology and Applications Symposium*, pp. 36-45, June 1999.

[6] Welch, L., Pfarr, B. and Tjaden, B., "Adaptive Resource Management Technology for Satellite Constellations," *The Second Earth Science Technology Conference (ESTC-2002)*, Pasadena, CA, June 2002.

[7] Welch, L., et al., "Adaptive Resource Management for On-board Image Processing Systems," *Journal. of Parallel & Distributed Computing Practices-* Special Issue on parallel & distributed real-time systems, Nova Science Publishers (to appear).

[8] von Neumann, J., and Morgenstern, O., "Theory of Games and Economic Behaviour", Princeton University Press, 1947

[9] Howard, R., and Matheson, J. eds., "*The Principles and Applications of Decision Analysis*", Strategic Decision Group, 1984

[10] Raiffa, H., and Kennez, R., "*Decisions with Multiple Objectives: Preferences and Value Tradeoffs*", Wiley, New Zork, 1976

[11] Otto, K., Antonsson, E., "*The Method of Imprecision Compared to Utility Theory for Design Selection Problems*", In Design Theory and Methodology (DTM93) ASME, pp. 167-173, 1993

[12] Ecker, K., Juedes, D., Welche, L., Chelberg, D., Bruggeman, C., Drews, F., Fleeman, D., Parrot, D., and Pfarr, B., "*An Optimization Framework for Dynamic, Distributed Real-Time Systems*", In Proceedings of the 11th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS2003), to appear, 2003

[13] Drews, F., Fleeman, D., Ecker, K., Welch, L, "*A General Optimization Framework and a System Architecture for Adative Resource Management*", Technical Report, Department of Computer Science, Ohio University, to appear, 2003.

[14] Blazewicz, J., Ecker, K., Schmidt, G., and Weglarz, J.,"*Schedulig in Computer and Manufacturing Systems*", Springer, 2001

[15] NASA SWIFT homepage, http://swift.gsfc.nasa.gov